

Projekt zaliczeniowy

JAVA - Programowanie Sieciowe



Grupa L5

Piotr hadała

ARKADIUSZ NOWAK

PAWEŁ JĘDRZEJCZAK

NATALIA GÓROWSKA

Spis treści

[1. Wstęp 3](#_Toc158126957)

[2. Użyte technologie 3](#_Toc158126958)

[3. Uruchomienie projektu 3](#_Toc158126959)

[4. Token 4](#_Toc158126960)

[Adres do Tokena 4](#_Toc158126961)

[5. BookApiController 4](#_Toc158126962)

[Adres do API 6](#_Toc158126963)

[Otrzymanie książki po ID 7](#_Toc158126964)

[Książki rekomendowane 7](#_Toc158126965)

[Dodawanie książek 8](#_Toc158126966)

[6. GenreApiController 9](#_Toc158126967)

[Adres do API 10](#_Toc158126968)

[Otrzymanie informacji o gatunku po jego nazwie 11](#_Toc158126969)

[Dodawanie informacji o gatunku książki 12](#_Toc158126970)

[7. Walidacja zapytań 13](#_Toc158126971)

[8. Baza danych 14](#_Toc158126972)

[master.xml 14](#_Toc158126973)

[001\_create\_table\_genre.xml 14](#_Toc158126974)

[001\_genres.xml 14](#_Toc158126975)

[002\_create\_table\_book.xml 14](#_Toc158126976)

[002\_books.xml 14](#_Toc158126977)

[003\_create\_tables\_users\_and\_roles.xml 14](#_Toc158126978)

[003\_users\_and\_roles.xml 15](#_Toc158126979)

[004\_create\_table\_book\_ratings.xml 15](#_Toc158126980)

[004\_book\_ratings.xml 15](#_Toc158126981)

[9. Podział na mikroserwisy 16](#_Toc158126982)

# Wstęp

Projekt robiony na ocenę 5.0. W zadaniach na ocenę 3.0 wykonano polecenia z wariantu numer 1.   
W tym pliku opisany znajduje się dokumentacja całego projektu. Są tu utworzone dwa API, jedno do zwracania danych na temat książek znajdujących się w bazie danych, a drugie natomiast zwraca informacje odnośnie gatunków książek.



# Użyte technologie

Apache Maven (wersja 4.0.0) <https://maven.apache.org/download.cgi>

Spring Boot (wersja 3.2.1) <https://start.spring.io>

# Uruchomienie projektu

Na samym początku musimy stworzyć pustą bazę w mysql, o nazwie **springlibrary**(można też nadać jej inną nazwę, ale wtedy trzeba ją też wpisać w pliku application.properties w folderze rest-service i web-service).

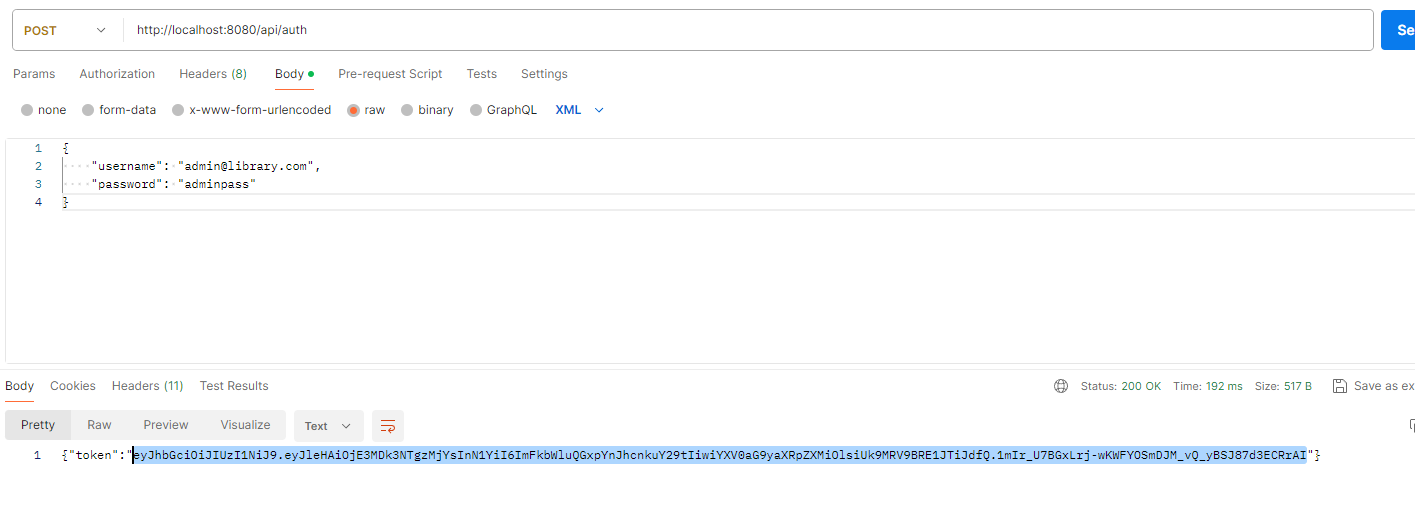
W katalogu głównym projektu widzimy cztery foldery: eureka-service, gateway-service, rest-service i web-service. Każdy z nich otwieramy jako oddzielny projekt, najlepiej w IntelliJ Idea. Uruchamiamy wszystkie service i możemy już wejść na naszą stronę, domyślnie wpisując w przeglądarkę http://localhost:8080/

# Token

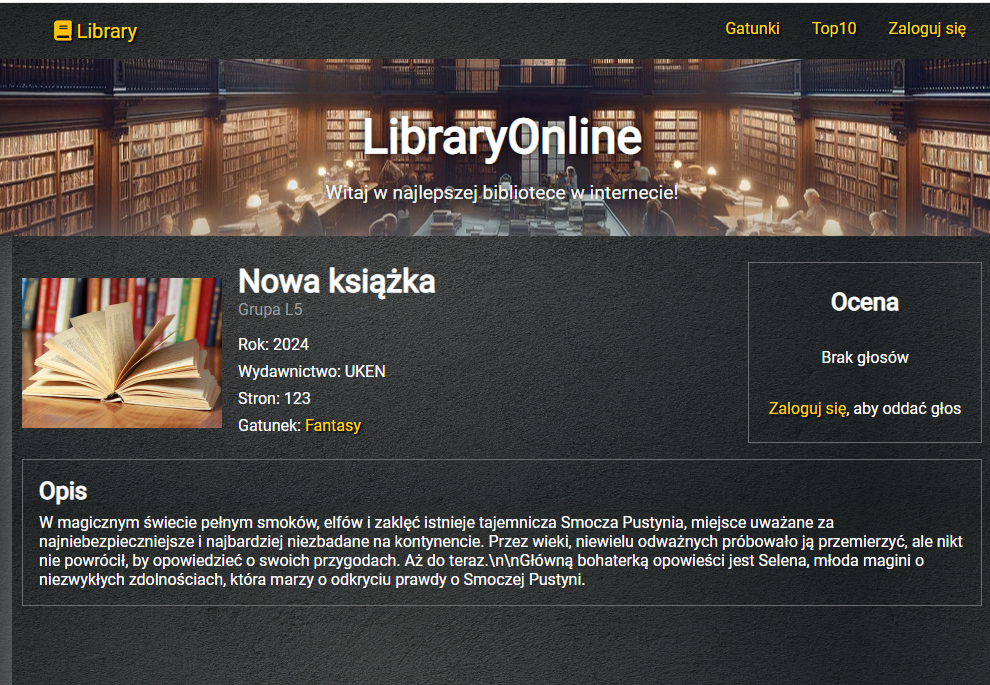
## Adres do Tokena

**GET /api/auth**

* **Parametry:** UserDto – obiekt reprezentujący użytkownika(username, password)
* **Zwracane wartości:** token



# BookApiController



Klasa BookApiController to kontroler, obsługujący zapytania dotyczące książek w API biblioteki.

Adnotacja @RestController określa, że jest to kontroler REST, który obsługuje zapytania HTTP.

Adnotacja @RequestMapping określa bazową ścieżkę dla wszystkich endpointów definiowanych w tej klasie.

Poniżej znajduje się opis poszczególnych endpointów tego kontrolera.

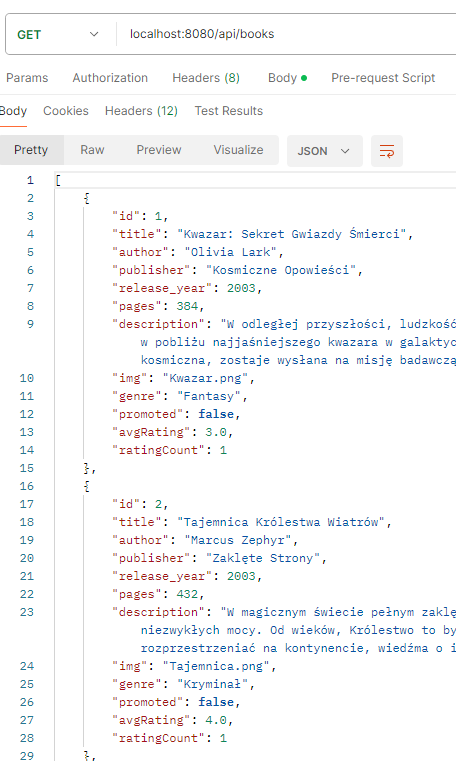
Plik BookApiController.java:

package pl.library.libraryonlineapiservice.web.api;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.server.ResponseStatusException;  
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;  
import pl.library.libraryonlineapiservice.domain.book.BookService;  
import pl.library.libraryonlineapiservice.domain.book.dto.BookDto;  
import pl.library.libraryonlineapiservice.domain.book.dto.BookSaveApiDto;  
  
import java.net.URI;  
import java.util.List;  
  
@RestController  
@RequestMapping(  
 path = "/api/books",  
 produces = {  
 MediaType.APPLICATION\_JSON\_VALUE,  
 MediaType.APPLICATION\_XML\_VALUE  
 })  
public class BookApiController {  
 private final BookService bookService;  
  
 public BookApiController(BookService bookService) {  
 this.bookService = bookService;  
 }  
  
 @GetMapping()  
 public List<BookDto> getAllBooks() {  
 return bookService.findAllBooks();  
 }  
  
 @GetMapping("/{id}")  
 public BookDto getSingleBook(@PathVariable long id) {  
 BookDto book = bookService.findBookById(id)  
 .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT\_FOUND));  
 return book;  
 }  
  
 @GetMapping("/recommended")  
 public List<BookDto> getRecommendedBooks() {  
 return bookService.findAllPromotedBooks();  
 }  
  
 @PostMapping  
 ResponseEntity<BookDto> addBook(@RequestBody BookSaveApiDto bookDto) {  
 BookDto addedBook = bookService.createBook(bookDto);  
 URI savedJobOfferUri = ServletUriComponentsBuilder.fromCurrentRequest()  
 .path("/{id}")  
 .buildAndExpand(addedBook.getId())  
 .toUri();  
 return ResponseEntity.created(savedJobOfferUri).body(addedBook);  
 }  
}

## Adres do API

**GET /api/books/**

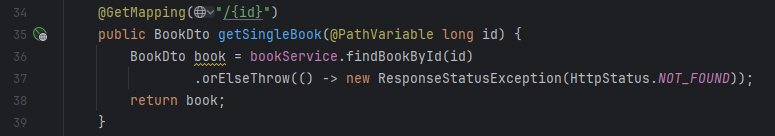
Aby otrzymać dane z API należy dodać do adresu ***/api/books*.** Otrzymamy wtedy listę wszystkich pozycji znajdujących się w bazie.



## Otrzymanie książki po ID

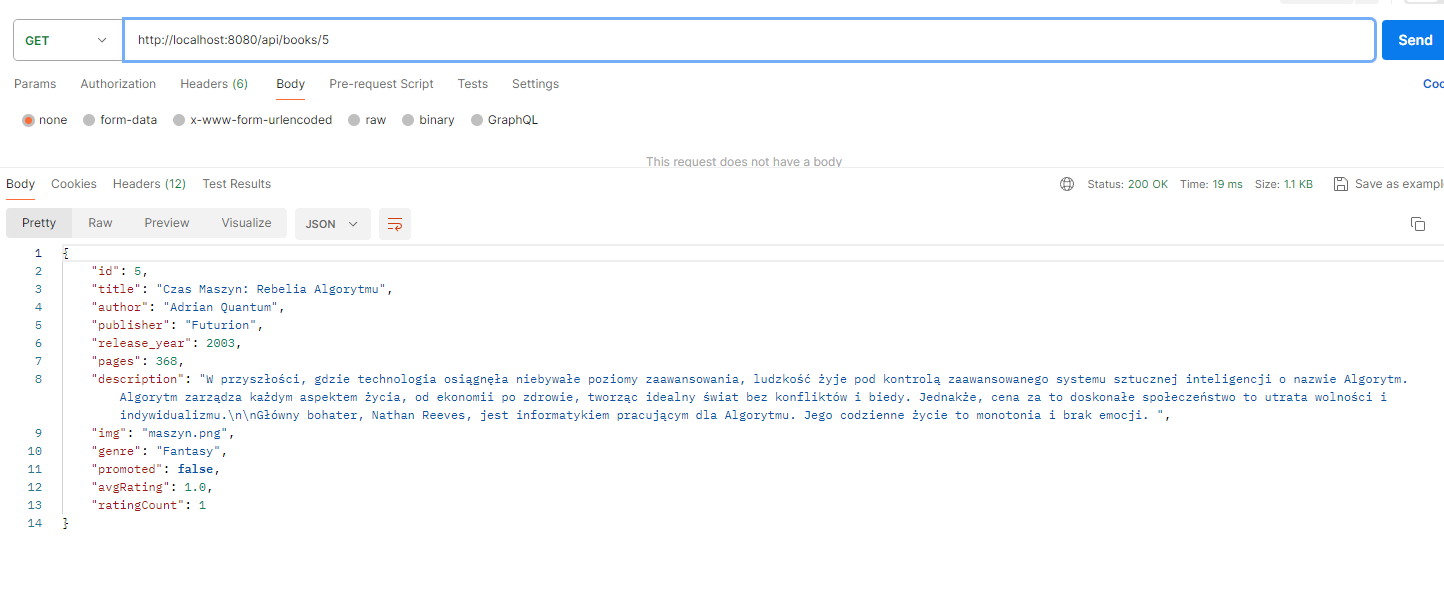
**GET /api/books/{id}/**

Po wpisaniu /api/books/{id}, np. **/api/books/5** otrzymamy informacje o pozycji z ID równym 5.



Zwraca pojedynczą ksiażkę o podanym id. Jeśli jej nie znaleziono, zwraca błąd 404 (NOT\_FOUND).

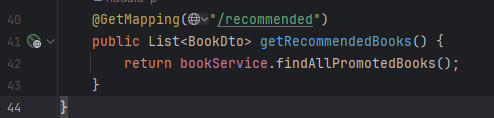
* **Parametry:** id – identyfikator książki
* **Zwracane wartości:** obiekt book, reprezentujący znalezioną książkę



## Książki rekomendowane

**GET /api/books/recommended**

Ścieżka **/api/books/recommended** zwróci nam pozycje, dla których wartość recommended wynosi true.



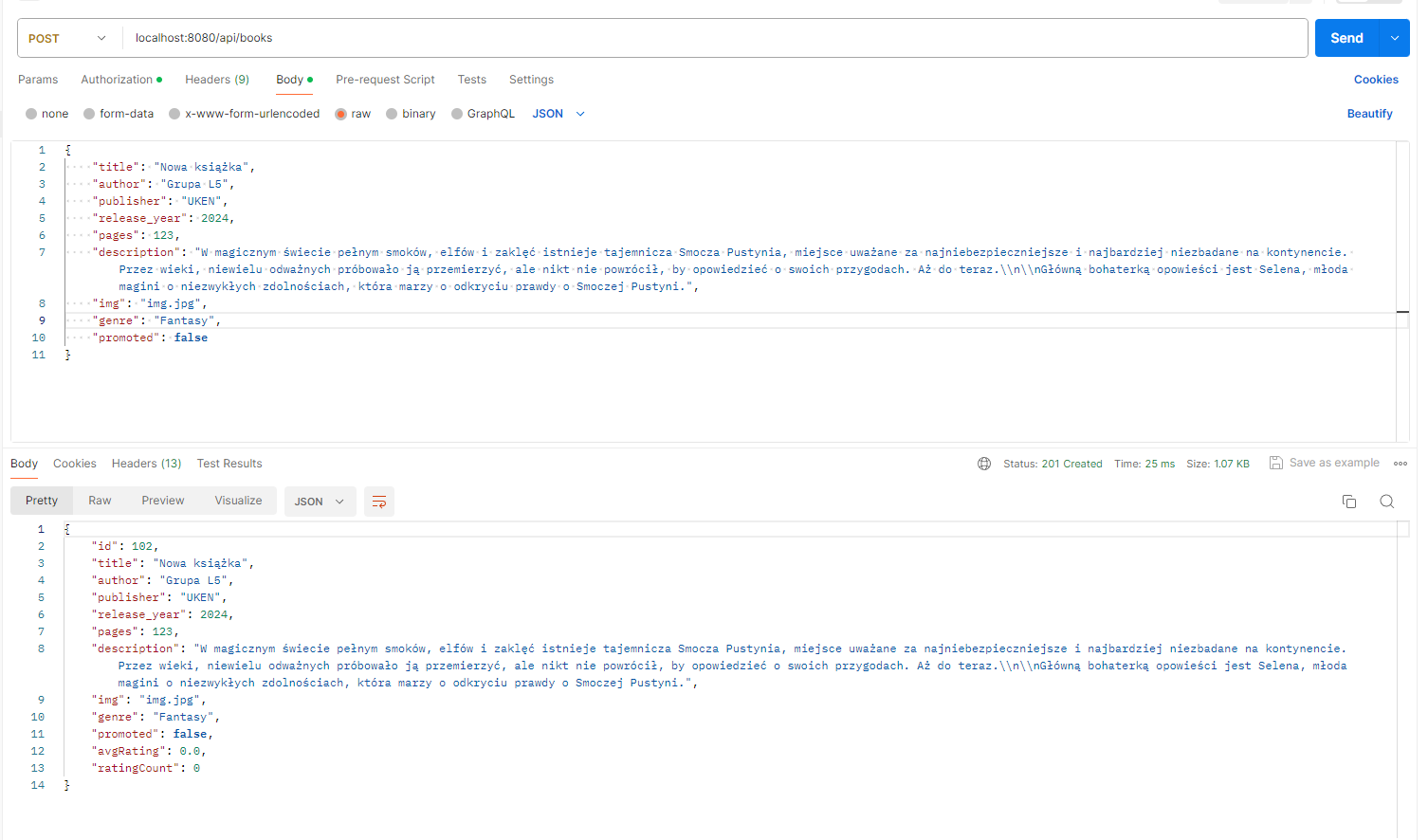


## Dodawanie książek

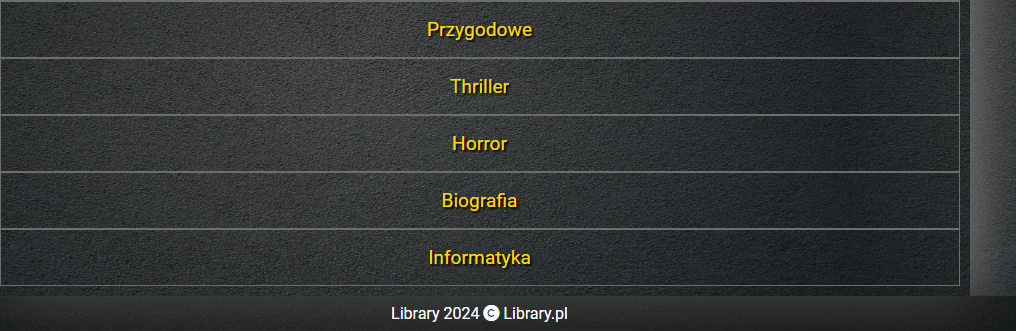
**POST /api/books/ addBook(BookSaveApiDto bookDto)**

Dodaje nową książkę do bazy danych.

* **Parametry: bookDto –** obiekt, reprezentujący książkę
* **Token**
* **Zwracane wartości:** kod 201 (created), jeśli dodano książkę



# GenreApiController



To API pozwoli nam na otrzymywanie informacji na gatunków książek które posiadamy w naszej bazie danych.

Plik GenreApiController.java:

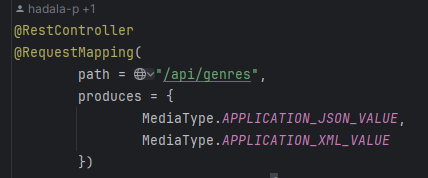
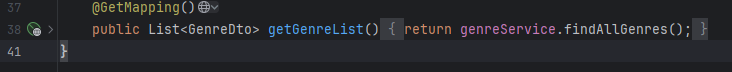
package pl.library.libraryonlineapiservice.web.api;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.server.ResponseStatusException;  
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;  
import pl.library.libraryonlineapiservice.domain.genre.GenreService;  
import pl.library.libraryonlineapiservice.domain.genre.dto.GenreDto;  
  
import java.net.URI;  
import java.util.List;  
  
@RestController  
@RequestMapping(  
 path = "/api/genres",  
 produces = {  
 MediaType.APPLICATION\_JSON\_VALUE,  
 MediaType.APPLICATION\_XML\_VALUE  
 })  
public class GenreApiController {  
 private final GenreService genreService;  
  
 public GenreApiController(GenreService genreService) {  
 this.genreService = genreService;  
 }  
  
 @GetMapping("/{name}")  
 public GenreDto getGenre(@PathVariable String name) {  
 GenreDto genre = genreService.findGenreByName(name)  
 .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT\_FOUND));  
 return genre;  
 }  
  
 @GetMapping  
 public List<GenreDto> getGenreList() {  
 return genreService.findAllGenres();  
 }  
 @PostMapping  
 ResponseEntity<GenreDto> addGenre(@RequestBody GenreDto genreDto) {  
 GenreDto addedGenre = genreService.addGenre(genreDto);  
 URI savedJobOfferUri = ServletUriComponentsBuilder.fromCurrentRequest()  
 .path("/{id}")  
 .buildAndExpand(addedGenre.getId())  
 .toUri();  
 return ResponseEntity.created(savedJobOfferUri).body(addedGenre);  
 }  
}

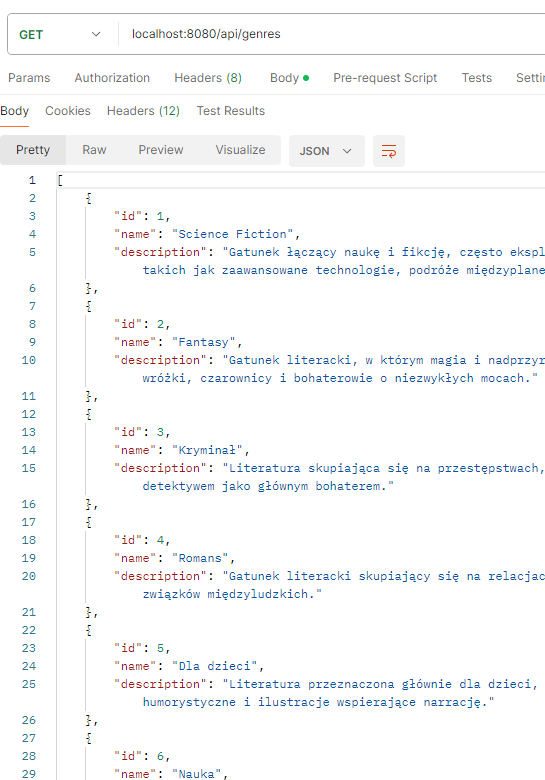
## Adres do API

**GET /api/genres/**

Wyszukuje wszystkie gatunki i zwraca je w postaci listy.

* **Zwracane wartości:** obiekt List<Genre> będący listą wszystkich znalezionych gatunków





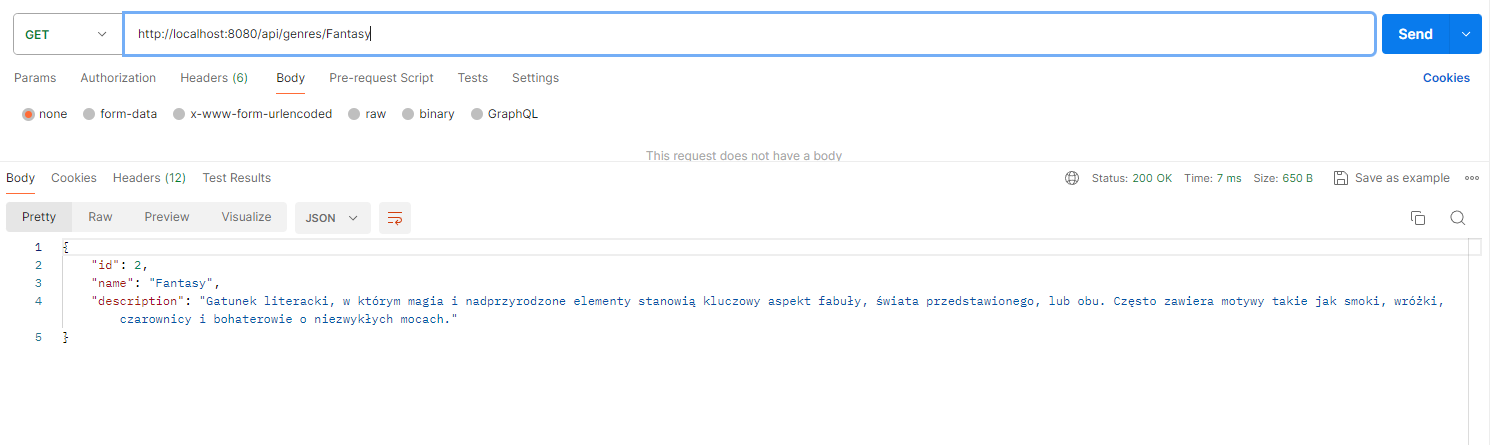
## Otrzymanie informacji o gatunku po jego nazwie

**GET /api/genres/{name}**

Zwraca informacje o gatunku na podstawie jego nazwy.

* **Parametry:** name – nazwa gatunku
* **Zwracane wartości:** obiekt <GenreDto>, zawierający informacje o znalezionym gatunku

Po wpisaniu /api/genres/{name}, np. **/api/genres/Fantasy** otrzymamy informacje związane z rekordem o nazwie Fantasy.

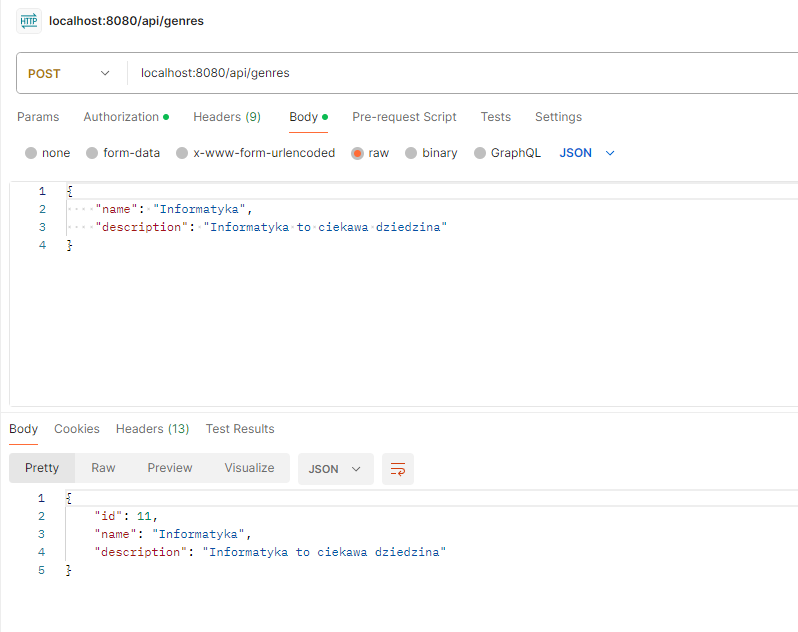


## Dodawanie informacji o gatunku książki

**POST /api/genres**

Dodaje nowy gatunek do bazy danych.

* **Parametry:** GenreDto – obiekt, reprezentujący gatunek
* **Token**
* **Zwracane wartości:** kod 201 (created), jeśli dodano gatunek

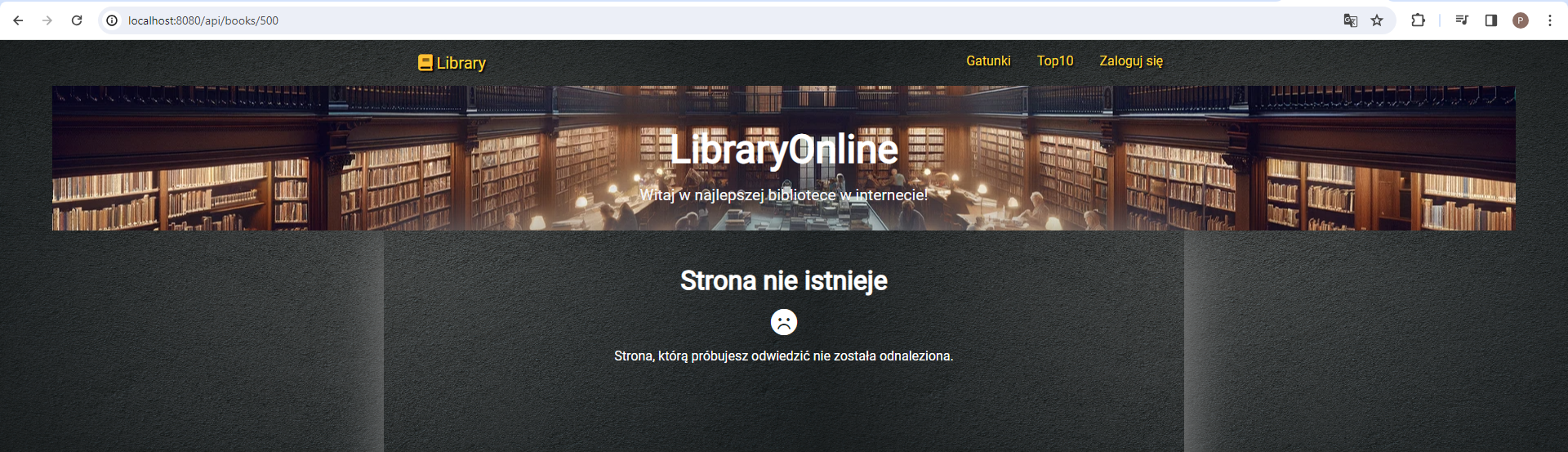


# Walidacja zapytań

Powyższe API zostały zabezpieczone przed nieprawidłowymi zapytaniami.



W przypadku gdy spróbujemy otrzymać dane o rekordzie, który nie istnieje, np. /api/books/500 (szukanie książki o id 500 – nie istnieje taka w bazie), to otrzymamy komunikat, że nie istnieje taka strona.



# Baza danych

## master.xml

Ten plik XML używa narzędzia Liquibase do zarządzania zmianami w strukturze bazy danych.

odnosi się do sytuacji, w jakiej wprowadzane są zmiany. W tym przypadku, zmiany danych testowych są stosowane tylko w środowisku oznaczonym jako "dev" - deweloperskie.  
  
Pliki create[...].xml odpowiadają za zdefiniowanie struktury baz danych - określają jak mają być tworzone bazy.

Pliki [...].xml określają, jakie dane mają znajdować się w poszczególnych bazach.

## 001\_create\_table\_genre.xml

Plik definiuje strukturę tabeli “genre” w bazie danych. Posiada ona następujące kolumny:

* id – ID gatunku książki
* name – nazwa gatunku, np. fantasy, romans
* description – opis gatunku

## 001\_genres.xml

Plik definiuje dane początkowe, które będą znajdować się w tabeli “genre”.

## 002\_create\_table\_book.xml

Plik definiuje strukturę tabeli “books” w bazie danych. Posiada ona następujące kolumny:

* title – ID książki
* author – autor książki
* publisher – wydawnictwo, które opublikowało książkę
* release\_year – rok wydania książki
* pages - ilość stron książki
* description – opis książki
* img – obraz, przedstawiający okładkę książki
* promoted - określa, czy książka jest promowana
* genre\_id – ID gatunku książki

## 002\_books.xml

Plik definiuje dane początkowe, które będą znajdować się w tabeli “books”.

## 003\_create\_tables\_users\_and\_roles.xml

Plik definiuje strukturę trzech powiązanych ze sobą tabel: "users", "user\_role", oraz "user\_roles" w bazie danych. Tabele te posiadają następujące kolumny:

* **users**
  + id – ID użytkownika
  + email – adres email użytkownika
  + password - hasło użytkownika
* **user\_role**
  + id – ID roli użytkownika
  + name – nazwa roli
  + description – opis roli
* **user\_roles**
  + user\_id – ID użytkownika
  + role\_id – ID roli

## 003\_users\_and\_roles.xml

Plik definiuje dane początkowe, które będą znajdować się w tabelach “users”, “user\_role” i “user\_roles”.

## 004\_create\_table\_book\_ratings.xml

Plik definiuje strukturę tabeli “book\_rating” w bazie danych. Posiada ona następujące kolumny:

* id – ID oceny
* user\_id – ID użytkownika, który ocenia książkę
* book\_id – ID ocenianej książki
* rating - wartość liczbowa oceny

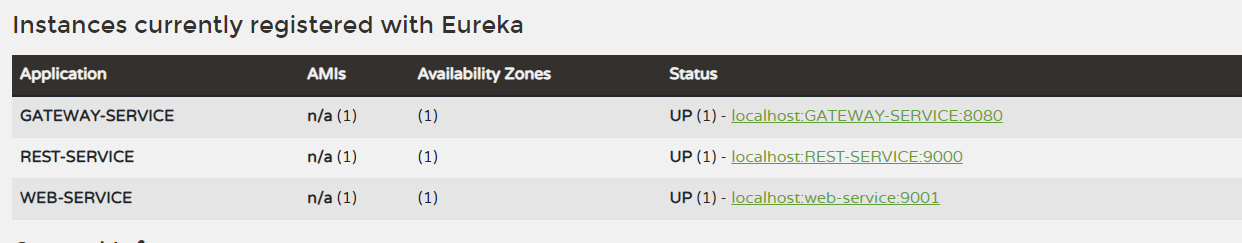
## 004\_book\_ratings.xml

Plik definiuje dane początkowe, które będą znajdować się w tabeli “book\_rating”.

# Podział na mikroserwisy

Aplikacja podzielona jest na mikroserwisy:

* eureka-service
* gateway-service
* rest-service
* web-service



Zawartość Rest-serivce:

**BookService.java**

Klasa odpowiada za zarządzanie książkami w systemie biblioteki. Poniżej znajduje się opis poszczególnych metod klasy.

### **findAllPromotedBooks()**

Zwraca listę wszystkich książek, oznaczonych jako promowane.

* **Zwracane wartości:** Lista promowanych książek- obiekt <BookDto>

### **findBookById(long id)**

Zwraca książkę z danym ID

* **Parametry**: id – ID książki
* **Zwracane wartości:** Obiekt Optional<BookDto>, zawierający znalezioną książkę.

### **findBooksByGenreName(String genre)**

Zwraca listę książek należących do określonego gatunku.

* **Parametry:** genre – nazwa gatunku
* **Zwracane wartości:** Lista książek danego gatunku - obiekt List<BookDto>

### **findAllBooks()**

Zwraca listę wszystkich książek dostępnych w systemie.

* **Zwracane wartości:** Lista wszystkich książek w formacie List<BookDto>

### **findTopBooks()**

Zwraca listę książek z najwyższymi ocenami.

* **Parametry:** size - ilość książek do znalezienia
* **Zwracane wartości:** Lista znalezionych książek w formacie List<BookDto>

### **addBook(BookSaveDto bookToSave)**

Dodaje nową książkę do bazy danych.

* **Parametry:** bookToSave – przechowuje wszystkie potrzebne dane o książce, którą chcemy dodać do systemu.

### **createBook(BookSaveApiDto bookToSave)**

Dodaje nową książkę do bazy danych.

* **Parametry:** bookToSave – przechowuje wszystkie potrzebne dane o książce, którą chcemy dodać do systemu.
* **Zwracane wartości:** Obiekt <BookDto> reprezentujący nową książkę.

**GenreService.java**

Klasa odpowiada za zarządzanie gatunkami książek w systemie.

### **FindGenreByName(String name)**

### Pobiera informacje o gatunku na podstawie jego nazwy.

* **Parametry:** name – nazwa gatunku
* **Zwracane wartości:** Obiekt Optional<GenreDto> zawierający informację o znalezionym gatunku

### **FindAllGenres()**

### Zwraca listę wszystkich znalezionych gatunków.

* **Zwracane wartości:** List<GenreDto> - lista wszystkich gatunków

### **addGenre(GenreDto genre)**

Dodaje nowy gatunek do bazy danych.

* **Parametry:** genre – informacje o nowym gatunku, który chcemy dodać
* **Zwracane wartości:** Obiekt GenreDto> zawierający informacje o dodanym gatunku

**RatingService.java**

Odpowiada za obsługę ocen książek, dodawanych przez użytkowników.

### **addOrUpdateRating(String userEmail, long bookId, int rating)**

### Dodaje lub aktualizuje ocenę, zamieszczoną przez użytkownika dla konkretnej książki.

* **Parametry:** 
  + userEmail – adres email użytkownika
  + bookId – ID ksiażki, którą ocenia użytkownik
  + rating - wartość oceny, przyznawanej książce

### **getUserRatingForBook(String userEmail, long bookId)**

Pobiera ocenę wystawioną przez użytkownika dla konkretnej książki.

* **Parametry:** 
  + userEmail – adres email użytkownika
  + bookId – ID ksiażki, którą ocenił użytkownik
* **Zwracane wartości:** Obiekt Optional<Integer> zawierający wartość oceny użytkownika dla danej książki

**UserService.java**

Klasa pozwala na zarządzanie użytkownikami biblioteki.

**findCredentialsByEmail(String email)**

Pobiera dane uwierzytelniające użytkownika na podstawie jego adresu email.

* **Parametry:** email – adres email użytkownika
* **Zwracane wartości:** email – Obiekt Optional<UserCredentialsDto> zawierający dane uwierzytelniające użytkownika

**registerUserWithDefaultRole(UserRegistrationDto userRegistration)**

Rejestruje nowego użytkownika, przypisując mu domyślną rolę.

* **Parametry:** Obiekt UserRegistrationDto zawierający dane potrzebne do rejestracji nowego użytkownika